CHAPTER 3. 파이썬 기초 자료형

생성자	선 선우 박
생성 일시	@2024년 8월 10일 오후 5:08
태그	인프런 파이썬

▼ 3-1. 숫자형

- 숫자형 사용법
  - 파이썬 모든 자료형

```
int : 정수
float : 실수
complex : 복소수
bool : 불린 (참/거짓)
str : 문자열(시퀀스)
list : 리스트(시퀀스)
tuple : 튜플(시퀀스)
dict : 딕셔너리
set : 집합
```

  - 데이터 타입 선언

```
# 데이터 타입
int = 7
float = 1.2
bool = True / False
str = ‘Python’
list = [‘Python’, ‘Study’]
tuple = (7, 8, 9) # tuple = 7, 8, 9도 가능
dict = {
        ‘name’ : ‘Machine Learning’,
        ‘version’ : 2.0
        }
set = {3, 5, 7}
```

```
# 데이터 타입 출력
print(type(str))
print(type(bool))
print(type(float))
print(type(int))
print(type(dict))
print(type(tuple))
print(type(set))
```

- 연산자 활용

```
# 숫자형 연산자
+
-
*
/
// : 몫
% : 나머지
abs(x) : 절대값
pow(x, y) : x**y, x의 y제곱

#예시
i1 = 39
i2 = 23
print(i1 + i2)
print(i1 * i2)
print(abs(-7))
>> 7
print(pow(5,3), 5 ** 3)
>> 125, 125
```

- 형 변환

```
# 형 변환 실습
a = 3. # 3.0
b = 6
c = .7 # 0.7
```

```
d = 12.7

# 타입 출력
print(type(a), type(b), type(c), type(d))

# 형 변환
print(float(b))
print(int(c))
print(int(d))
print(int(True)) # True : 1, False : 0
print(float(False))
print(complex(3))
print(complex('3')) # 문자형 -> 숫자형
print(complex(False))

# divmod : x를 y로 나누었을 때 몫과 나머지를 튜플로 반환
# 예시
x, y = divmod(100, 8)
print(x, y)
>> (12, 4)
```

- 외부 모듈 사용

```
> import math

print(math.ceil(5.1)) # x 이상의 수 중에서 가장 적은 정수
>> 6
print(math.pi)
>> 3.141592653589793
```

▼ 3-2. 문자형

- 문자형 사용법
  - 문자열 생성

```
str1 = "I am Python"
str2 = 'python'
str3 = """How are you?"""
str4 = '''Thank you!'''
```

```
print(len(str1), len(str2), len(str3), len(str4))
```

- 빈 문자열

```
# 빈 문자열
str1_t1 = ''
str2_t2 = str()
```

- 이스케이프

```
'''
참고 : Escape 코드
\n : 개행
\t : 탭
\\ : 문자
\' : 문자
\" : 문자
\000 : 널 문자
'''

# 이스케이프 문자 사용
# I'm Boy

print("I'm Boy")
>> I'm Boy
print('I\'m Boy')
>> I'm Boy
print('I\\m Boy')
>> I\m Boy

print('a \t b') # 탭
>> a    b
print('a \n b') # 줄바꿈
>> a
b
print(' a\"\"b') # ""출력
>> a""b
```

```
# Raw String
raw_s1 = r'D:\tpython\test'
print(raw_s1)
>> D:\tpython\test
```

- 멀티 라인

```
# 멀티라인 입력
# 역슬래시 사용하면 오류 X

multi_str = \
'''
문자열
멀티라인 입력
테스트
'''

print(multi_str)
```

- 문자형 연산

```
# 문자열 연산
str_o1 = "Python"
str_o2 = "Apple"
str_o3 = "How are you doing"
str_o4 = "Seoul Dajeon Busan Jinju"

print(str_o1 * 3)
>> PythonPythonPython
print(str_o1 + str_o2)
>> PythonApple
print('y' in str_o1) # y가 포함되어있는지
>> True
print('P' not in str_o2) # P가 포함되어있지 않음
>> False
```

- 문자형 형 변환

```
# 문자열 형 변환
print(str(66), type(str(66)))
>> 66 <class 'str'>
print(str(10.1), type(str(10.1)
>> 10.1 <class 'str'>
print(str(True), type(str(True)))
>> True <class 'str'>
```

- 문자열 함수

```
# 문자열 함수(upper, isalunum, startswith, count, endswi
th, isalpha..)
print(str_o1.capitalize()) # 첫 글자를 대문자로
>> Python
print(str_o2.endswith("e")) # e로 끝나는지, 마지막문자확인
>> True
print(str_o1.replace("thon", "Good")) # 첫번째 부분을 찾
아서 두번째 부분으로 바꾸기
>> PyGood
print(sorted(str_o1)) # 정렬
>> ['P', 'h', 'n', 'o', 't', 'y']
print(str_o4.split(' ')) # ' ' 공백 기준으로 나누기
>> ['Seoul', 'Dajeon', 'Busan', 'Jinju']
```

- 슬라이싱

```
# 슬라이싱
str_s1 = "Nice Python"

print(len(str_s1))
>> 11

# 슬라이싱 연습
print(str_s1[0:3]) #3-1까지 나옴 0 1 2
>> Nic
print(str_s1[5:]) # [5:11] #Python 출력
>> Python
```

```
print(str_s1[:len(str_s1)]) # str_s1[:11]
>> Nice Python
print(str_s1[:len(str_s1)-1]) # str_s1[:10]
>> Nice Pytho
print(str_s1[1:4:2]) # 1부터 4까지 2칸 단위로
>> ie
print(str_s1[-5:]) # 뒤에서 5번째부터 오른쪽으로 진행
>> ython
print(str_s1[1:-2]) # ice Pyth
>> ice Pyth
print(str_s1[::2]) # 처음부터 끝까지 2칸씩
>> Nc yhn
print(str_s1[::-1]) # 맨마지막부터 역순으로
>> nohtyP eciN
```

▼ 3-3. 리스트

- 리스트 사용법
  - 리스트 선언

```
a = []
b = list()
c = [70, 75, 80, 85] # Len
d = [1000, 10000, 'Ace', 'Base', 'Captine']
e = [1000, 10000, 'Ace', ['Base', 'Captine']]
f = [21, 42, 'footbar', 3, 4, False, 3.14159]
```

  - 리스트 특징

```
# 자료구조에서 중요
# 리스트 자료형(순서O, 중복O, 수정O, 삭제O)
```

  - 리스트 인덱싱

```
# 인덱싱 (원하는 데이터를 꺼내오는 과정)
d = [1000, 10000, 'Ace', 'Base', 'Captine']
e = [1000, 10000, 'Ace', ['Base', 'Captine']]
```

```
print(type(d), d)
>> <class 'list'> [1000, 10000, 'Ace', 'Base', 'Capti
ne']
print(d[1])
>> 10000
print(d[0] + d[1] + d[1])
>> 21000
print(d[-1]) # Cpatine
>> Captine
print(e[-1][1])
>> Captine
print(list(e[-1][1]))
>> ['C', 'a', 'p', 't', 'i', 'n', 'e']
```

- 리스트 슬라이싱

```
# 슬라이싱
print(d[0:3])
>> [1000, 10000, 'Ace']
print(d[2:])
>> ['Ace', 'Base', 'Captine']
print(e[-1][1:3])
>> ['Captine']
```

- 리스트 연산

```
# 리스트 연산
print('c + d = ',c + d)
print('c * 3 = ', c * 3)
print("'Test' + c[0] = ", 'Test' + str(c[0]))

# 값 비교
print(c == c[:3] + c[3:])
print(c)
print(c[:3] + c[3:])
print()
```

- 리스트 함수

```
# 리스트 함수
a = [5, 2, 3, 1, 4]

a.append(10) # 끝부분에 데이터 삽입
a.sort() # 오름차순으로 정리
a.reverse() # 역
a.index(3) # =a[3] 인덱스3 3번째자리
a.insert(2, 7) # 2번째 위치에 7을 삽입
print(a.count(4)) # 리스트에 4가 몇개 있는지

ex = [8, 9]
a.extend(ex) # a 뒤에 ex 추가
print('a = ', a)

# 삭제 : remove, del, pop
# del은 인덱스로, remove는 값으로 찾아서 지움
# remove
a.remove(10) # 10을 제거
# del
del a[2]
# pop
print(a.pop()) # 리스트에서 마지막 원소를 선택한 다음 지움
print(a) # 마지막 원소 지워짐

# 반복문 활용
while a:
    data = a.pop()
    print(data)
```

- 리스트 수정, 삭제

```
# 리스트 수정
c = [70, 75, 80, 85]

c[0] = 4 # 0번째 인덱스의 값을 4로 수정
>> [4, 75, 80, 85]

c[1:2] = ['a', 'b', 'c'] # 범위 지정 -> 원소로 들어감
```

```
>> [4, 'a', 'b', 'c', 80, 85]

c[1] = ['a', 'b', 'c'] # 자리 지정 -> 리스트로 들어감
>> [4, ['a', 'b', 'c'], 'b', 'c', 80, 85]

c[1:3] = [] # 1번째 인덱스부터 2번째 인덱스까지의 값을 지움
>> [4, 'c', 80, 85]
```

▼ 3-4 튜플

- 튜플 사용법
  - 튜플 선언

```
# 선언
a = ()
b = (1,) # 원소가 하나일 땐 꼭 ,를 붙여야 튜플로 인식함
c = (11, 12, 13, 14)
d = (100, 1000, 'Ace', 'Base', 'Captine')
e = (100, 1000, ('Ace', 'Base', 'Captine'))
```

  - 튜플 특징

```
# 리스트와 비교 중요
# 튜플 자료형 (순서O, 중복O, 수정X, 삭제X) # 불변, 바뀌면 안되는 정보 입력하면좋음
```

  - 튜플 인덱싱

```
# 인덱싱
print( d[1])
print(d[0] + d[1] + d[1])
print(d[-1])
print(e[-1])
print(e[-1][1])
print(list(e[-1][1]))

# 수정X
# d[0] = 1500 -> 오류
```

- 튜플 슬라이싱

```
# 슬라이싱
print(d[0:3])
print(d[2:])
print(e[2][1:3])
```

- 튜플 함수

```
# 튜플 함수
a = (5, 2, 3, 1, 4)
print(a)

# 3이 몇 번 인덱스에 들어있는지
print(a.index(3))

# 2가 몇번 들어가있는지
print(a.count(2))
```

- 팩킹 & 언팩킹

```
# 팩킹
# 하나로 묶는 것
t = ('foo', 'bar', 'baz', 'qux')

print(t)
print(t[0])
print(t[-1])

# 언팩킹1
# 하나로 되어 있던 튜플을 풀어서 각각 할당해 주는 것
# 괄호 없이도 가능
(x1, x2, x3, x4) = t

print(type(x1), type(x2), type(x3), type(x4))
print(x1, x2, x3, x4)

# 팩킹 & 언팩킹
```

```
t2 = 1, 2, 3
t3 = 4
x1, x2, x3, x4 = t2
x4, x5, x6 = 4, 5, 6

print(t2)
print(t3)
print(x1, x2, x3)
print(x4, x5, x6)
```

▼ 3-5 딕셔너리

- 딕셔너리 사용법
  - 딕셔너리 선언

```
# 선언
# { 키 : 값 }
a = {'name' : 'Kim', 'phone' : '01012345678'}
b = {0: 'Hello Python'}
c = {'arr' : [1, 2, 3, 4]}
d = {
    'Name' : 'Niceman',
    'City' : 'Seoul',
    'Age' : 33,
}

e = dict([
    ('Name', 'Niceman'),
    ('City', 'Seoul'),
    ('Age', 33),
])

f = dict(
    Name = 'Niceman',
    City = 'Seoul',
    Age = 33,
)

# 출력
```

```
print('a - ', a['name']) # 키 존재X -> 에러
print('a - ', a.get('name')) # 키 존재X -> None 처리
print('b - ', b[0])
print('b - ', b.get(0))
print('f - ', f.get('City'))
print('f - ', f.get('Age'))
```

- 딕셔너리 특징

```
# 범용적으로 가장 많이 사용
# 딕셔너리 자료형(순서X, 키 중복X, 수정O, 삭제O)
```

- 딕셔너리 수정

```
# 수정
f.update(Age=36)

temp = {'Age': 27}
f.update(temp)

# 딕셔너리 추가
a['adress'] = 'seoul'
a['rank'] = [1, 2, 3]
```

- 딕셔너리 함수

```
# dict_keys, dict_values, dict_items : 반복문(__iter__)
에서 사용 가능
# 키만 출력
print(a.keys())

# 키를 리스트 형식으로 출력
print(list(a.keys()))

# 값만 출력
print(a.values())

# 값을 리스트 형식으로 출력
```

```
print(list(a.values()))

# 키와 값을 한번에 출력
print(a.items())

# 키와 값을 리스트로
print(list(a.items()))

# 예외
# print('f - ', f.popitem())
# True, False
print('name' in a)
print('addr' in a)
```

▼ 3-6 집합

- 집합 선언

```
# 선언
a = set()
b = set([1, 2, 3, 4])
c = set([1, 4, 5, 6])
d = set([1, 2, 'Pen', 'Cap', 'Plate'])
e = {'foo', 'bar', 'baz', 'foo', 'qux'}
f = {42, 'foo', (1, 2, 3), 3.14159}
```

- 집합 특징

```
#순서X, 중복X
```

- 튜플 변환

```
# 튜플 변환
t = tuple(b)
print(type(t), t)
print(t[0], t[1:3])
```

- 리스트 변환

```
# 리스트 변환
l = list(c)
l2 = list(e)
print(type(l), l)
print(l[0], l[1:3])
print(type(l2), l2)
```

- 집합 자료형 활용

```
# 집합 자료형 활용
s1 = set([1, 2, 3, 4, 5, 6])
s2 = set([4, 5, 6, 7, 8, 9])

# 교집합
print(s1 & s2)
print(s1.intersection(s2))

# 합집합
print(s1 | s2)
print(s1.union(s2))

# 차집합
print(s1 - s2)
print(s1.difference(s2))

# 중복 원소 확인
print(s1.isdisjoint(s2))

# 부분 집합 확인
print(s1.issubset(s2))
print(s1.issuperset(s2))
```

- 수정

```
# 추가 & 제거
s1 = set([1, 2, 3, 4])
```

```
s1.add(5)
print(s1)

s1.remove(2)
print(s1)
# s1.remove(7)
# 없는 원소 삭제하려고 하면 오류, discard는 오류 발생 X

s1.discard(3)
print(s1)

# 모두 제거
s1.clear()
```